Fall 2019

MFAProxy: A reverse proxy for multi-factor authentication

Alan Schmitz
aschmitz@iastate.edu

MFAProxy: A reverse proxy for multi-factor authentication

by

Alan Schmitz

A Creative Component submitted to the graduate faculty

in partial fulfillment of the requirements for the degree of

MASTER OF SCIENCE

Major: Information Assurance

Program of Study Committee:
Doug Jacobson, Major Professor

The student author, whose presentation of the scholarship herein was approved by the program of study committee, is solely responsible for the content of this Creative Component. The Graduate College will ensure this Creative Component is globally accessible and will not permit alterations after a degree is conferred.

Iowa State University

Ames, Iowa

2019

TABLE OF CONTENTS

Page

LIST OF FIGURES

ABSTRACT

Multi-factor authentication has been shown to be an effective method to reduce the risk of remote attacks, because it prevents many attackers from easily gaining an initial foothold into an organization. Many sites only support single factor authentication based on passwords which have well known weaknesses. This paper describes MFAProxy, a reverse proxy that adds multi-factor authentication to sites that currently do not support it.

The proxy can be deployed in a variety of configurations within a network to meet specific security goals. It supports flexible combinations of several factors including passwords, one-time passwords, and tokens based on public-key cryptography. Each of these factors offer a unique balance of security and usability that must be considered when an organization deploys multi-factor authentication.

CHAPTER 1. INTRODUCTION

Most sites today rely on a username and password for authentication, even though the risks of passwords are widely known and well understood Yıldırım and Mackie (2019). On their own, users have a tendency to pick passwords that are easy for them to remember, which also makes them easy to guess. This can be mitigated by enforcing a minimum length and preventing users from choosing common words or phrases. Strict enforcement of strong passwords is not enough to make passwords secure on their own.

Even when an organization enforces a strong password policy, users may reuse them on multiple sites. When that happens any organization relying solely on passwords for security is only as secure as the weakest site where that password was used. Sites like HaveIBeenPwned.com Hunt (2019) track hundreds of millions of credentials that have been obtained from data breaches at compromised sites. Attackers will certainly try those credentials against an organization's sites as well.

One of the most common and effective attacks over the last several years have been phishing attacks. Phishing is a fraudulent attempt to obtain authentication credentials and other sensitive information by disguising a site as something trustworthy. It is often carried out through email spoofing, and it directs users to enter their credentials at a fake site that matches the appearance of a legitimate site. Even the most secure passwords are vulnerable to phishing.

Passwords alone only provide a single factor to authenticate a user. In response to the inherent limitations of passwords, organizations have added support for two-factor and multi-factor authentication to their sites. Supporting additional authentication factors can be a challenge for software or services based on specific network protocols like SMTP or IMAP, because the authentication methods are defined within the protocols themselves. In the case of sites using the HTTP or HTTPS protocols, it is easier to add support for multiple authentication factors, because access to

those sites can be wrapped in an additional layer of security that is presented directly to the user through an interactive interface in their web browser.

This project introduces MFAProxy, a reverse proxy server for the HTTPS protocol. MFAProxy can be inserted into the network between the end user's browser and one or more upstream web servers. It can enforce multiple authentication factors of its own, before the user is allowed to access the upstream site. This enables an organization to add support for multi-factor authentication to sites that do not currently support it without modifying the code or configuration of the original web applications.

To start, section 2 introduces some of the factors that are widely used in multi-factor authentication. The specific factors included in MFAProxy and the implementation details of the reverse proxy are described in section 3. Deployment considerations are presented in section 4 including two possible network configurations for the proxy and password and token deployment options. Section 5 covers potential weaknesses in the project. Finally, section 6 provides the reader with future work that could make the project more useful in a production environment, while section 7 provides some concluding thoughts.

CHAPTER 2. BACKGROUND

Multi-factor authentication is an authentication method in which a user is granted access only after successfully presenting multiple pieces of evidence to an authentication mechanism. Each piece of evidence is considered a factor. These factors may include something the user knows, something the user has, or something the user is. Multi-factor authentication relies on a combination of at least two types of factors to authenticate to a system.

2.1 Passwords and PINs

Passwords are an example of something the user knows. They are the most common type of knowledge factor, and they require that the user proves that they know a secret in order to authenticate. Traditionally, users have been expected to memorize their passwords. Today the use of password managers is widely encouraged, because they enable users to maintain complex and unique passwords without reusing the same password on multiple sites Constantin (2019).

The password is a secret word or string of characters that is used for user authentication. It is a shared secret between the user and the site that wants to authenticate them. Sites should not store passwords in plain text. If an attacker gains access to a plain text password store, all of the passwords in the store will be compromised. Instead passwords should be salted and hashed before they are stored. When a user enters a password on such a site, the password verification software runs through a cryptographic hash algorithm. If the hash value generated from the user's entry matches the hash in the password store, the factor is validated. Salts on the stored passwords reduce the risk of password cracking through dictionary and rainbow-table attacks.

A Personal Identification Number (PIN) is another example of something the user knows. Traditionally, these have been considered short numeric passwords implemented as a shared secret. In the context of multi-factor authentication a PIN is notably different than a password, because it is

not a shared secret. A multi-factor PIN is used to unlock something the user has. The site does not store the PIN in any form, and no form of the PIN is ever transmitted to the site as part of the authentication process. An example of a multi-factor authentication PIN would be a swipe pattern used to unlock an authentication app on a mobile device.

It can be difficult for a site to require multi-factor PINs, because their use must be enforced by a remote device. The use of PINs is covered in more detail in the section on WebAuthn, which supports authenticating devices as well as users.

2.2 Short Message Service

The use of codes transmitted over the short message service (SMS) is an example of something the user has. The user registers their mobile telephone number with a site. When the user wants to authenticate, the site generates a unique code and sends it to the user's mobile device via text message. The user enters the code into the site to validate the factor.

The mobile device is an example of something the user has. It is considered a second factor, because the code is transmitted out of band over the switched telephone network rather than the Internet. It is widely used as a second factor, because it is easy for sites to implement, and it only requires a mobile device with text message service, which many users already have and understand.

From a security standpoint SMS is not a reliable second factor Akers (2019). It was never intended to be used for security purposes, so many mobile devices show the contents of messages in lock screen notifications. Apps running on the mobile device, including possible Trojans, may be able to intercept text messages. The SIM card in the mobile device can be stolen and swapped into another device. SIM cards can be cloned to take over a victim's mobile number. Finally there are flaws in the SS7 protocol used by the switched telephone network that transmits the text messages. These are not theoretical attacks. All of these methods have been used to steal SMS authentication codes in practice.

Even though SMS is widely deployed and easy to use, support for SMS codes is not included in MFAProxy. The factor simply is not very secure. It is also the only factor that requires an account

with a third-party service provider to transmit authentication codes. Most SMS providers charge a fee per message as well.

2.3 One-Time Passwords

A one-time password (OTP) is a password that is only valid for a single session or transaction. Unlike static passwords, OTPs are not supposed to be vulnerable to replay attacks. They also avoid problems with password reuse across multiple sites. Even if an attacker captures the password and the OTP used on one site, the same factors will not work on another site, because the OTP will no longer be valid.

One-time passwords are difficult for humans to calculate, so they require additional technology. This technology makes OTPs an example of something the user has rather than something the user knows. OTPs are typically generated by a hardware token or an app running on a mobile device. Throughout the rest of this paper, this hardware or software that the user must possess will be referred to as a token.

Two examples of OTPs that have been standardized and widely implemented are HMAC-based one-time passwords (HOTP) and time-based one-time passwords (TOTP). Both of these methods rely on common parameters. They both use a cryptographic hash function, which defaults to SHA-1. They both need a shared secret key for input, which is supposed to be a minimum of 128 bits, but in practice is often only 80 bits. Finally, they both use a fixed output length to limit the number of digits required for the authentication code, which typically defaults to six digits.

HOTP is defined in RFC 4226 M'Raihi et al. (2005). In addition to the shared secret, it requires a counter as an input. Both the user and the site independently calculate the authentication code based on their local copy of the counter. If they match, the HOTP factor is validated, and the site saves its copy of the new counter value.

The user may increment their counter independently of the site, so the user's counter may be larger than the server's counter. To account for this possibility, the site tests a range of values larger than the current counter. If any of these codes can be validated, its corresponding counter

value is used to determine the value the server should store. As long as the user's counter does not exceed the value included in the range of values the site tests, the user's counter and the site's counter will stay synchronized.

In some cases it is possible for the site to detect that a user's HOTP authenticator has been cloned. The user's counter should always increase, so an authentication code based on a lower counter value should never be used again. If a lower counter value is reused, it indicates that there may be two or more copies of the shared secret in circulation with independent counters. Previous values are being reused, because the counter in one of the tokens falls behind the counter in another token. The server can test for this by also testing a range of counter values lower than its current counter value. These codes would not validate the HOTP factor, but the event could be logged as a possible cloned HOTP token by the site.

TOTP is defined in RFC 6238 M'Raihi et al. (2011). In addition to the shared secret, it requires the current time as an input. The current time is measured as the Unix time. It also uses an interval value which defaults to 30 seconds. The interval determines the length of time that an authentication code will be valid. Both the user and the site independently calculate the authentication code based on their local clocks. If they match, the TOTP factor is validated.

In order for TOTP authentication to function properly, the user's clock and the site's clock must be roughly synchronized. Most sites will validate the authentication codes for the next and previous time intervals as well as the current time interval. This helps alleviate problems caused by clock skew and delays while the user enters the code from their token into the site.

TOTP codes are often referred to as Google Authenticator codes, because they were popularized by the mobile app of the same name. Google supports their mobile app as an option for their own two-step verification account security. Even though this app uses defaults that are weaker than the values required by RFC 6238, other sites and OTP tokens now use the same defaults.

2.4 U2F and WebAuthn

Universal Second Factor (U2F) and WebAuthn are a series of standards that were first developed by Google and Yubico and later hosted by the FIDO Alliance (2019). They define protocols and interfaces for hardware tokens, so they are an example of something the user has. Current tokens communicate over USB or NFC, but standards are being developed to allow tokens to communicate over Bluetooth as well.

U2F is the original standard which is also called FIDO. It supports challenge response authentication using public-key cryptography. It is capable of providing a single authentication factor, so it must be used with another factor like a password to provide multi-factor authentication. It is resistant to site impersonation, which means it is resistant to active man-in-the-middle attacks. The private key stored on the token is not accessible by the host system, so it is also resistant to malware.

WebAuthn is a later standard, which is also called FIDO2. It is fully backward compatible with U2F / FIDO, and it is based on the same challenge response authentication using public keys. The primary difference between U2F and WebAuthn is that a WebAuthn token can also be used for multi-factor authentication on its own.

When used as a single factor, a WebAuthn token operates like an older U2F token and simply checks for the user's presence though a button press on the token. When a token operates in two-factor mode supported by FIDO2, it can verify something you know like a PIN, or something you are like a fingerprint. The PIN or biometric data is never sent to the site for validation. It is used locally by the token to unlock the token itself. The site does not simply trust any token to require a second factor. A WebAuthn token must also authenticate itself to the site. To address privacy concerns, the token only authenticates its manufacturer and model number, rather than a unique device identifier. This is called device attestation, and it enables a site to ensure that a specific make and model of device will enforce multi-factor authentication locally.

WebAuthn with device attestation enables some vendors to support truly password-less multi-factor authentication. The user must unlock a token with a PIN or a fingerprint. The token

then validates its make and model as well as the user that wants to authenticate with the site. Passwords are never transmitted to the site. Currently support for multi-factor WebAuthn is extremely limited, because there is no public-key infrastructure in place to a attest a large number of devices from a wide variety of manufacturers. Vendors tend to only support specific combinations of software and hardware tokens.

U2F and WebAuthn both maintain a counter like HOTP. U2F and WebAuthn are based on public-key cryptography rather than cryptographic hashes, so the counter is not required for the challenge-response protocol. It is only used to detect a possible cloned token.

MFAProxy does not support device attestation at this time, so it uses the WebAuthn protocol in its older form that is backward compatible with U2F and FIDO. Until a trusted certificate authority is available for FIDO2 compliant devices, the public keys required for device attestation would need to be hard coded into the software that supports it.

CHAPTER 3. IMPLEMENTATION

MFAProxy is a reverse HTTP proxy written in Go. This language was selected, because other popular servers that can be used as reverse HTTP proxies are written in Go, including Caddy and Traefik. The net/http module included in the Go standard library uses concurrency for performance, and it has native support for TLS. The standard library also includes a reverse proxy module that covers most of the HTTP protocol specification.

The proxy listens for incoming HTTPS requests and matches them against a list of patterns in the proxy's configuration. Each pattern includes a host name and URL path to match and an upstream server that will process the matching requests. Each pattern also includes a set of timers for any required authentication factors. The supported factors include password, HOTP, TOTP, and WebAuthn. A typical pattern will require a password factor as something the user knows and at least one of the HOTP, TOTP, or WebAuthn factors for something the user has.

The configuration requires at least two host names names for the proxy. One is reserved for the proxy itself. All authentication requests enforced by the proxy are required to use this host name. Each site that will be forwarded through the proxy requires a unique host name for the the proxy as well. Because the proxy only listens for requests over HTTPS, it is assumed that it will be configured with a wildcard TLS certificate, and all of the host names used will match the pattern defined in the common name of this certificate.

Any successful authentication of a factor by the proxy establishes a session with the proxy. Sessions are maintained using the SCS package Edwards (2019). This package follows the OWASP security guidelines for session management Foundation (2019). The TLS encryption provided by HTTPS is required to protect the cookie used by the session manager to reduce the risk of session hijacking. The session cookie is renewed after every successful authentication by the proxy to reduce the risk of session fixation. The SCS session manager enforces an inactivity timer and total

lifetime on every session. These default to 1 hour of inactivity and 1 day of total session time. The defaults can be overridden using settings in the proxy's configuration file.

The proxy maintains a timestamp in the user's session for each authentication factor. The timestamp is compared to the authentication timer defined in the pattern that matches the proxy's request. If the current time has not exceeded the time stamp in the session added to the corresponding authentication timer, the request is considered authenticated for the required factor. If the request is not currently authenticated for a required factor, the proxy presents the user with an authentication form for that factor instead. The authentication timers for the factors do not renew. The user must either wait for the timer to expire or manually logout of the proxy and then log back in again to refresh the timer.

When the proxy interrupts the user's request for a web page to request authentication, it saves the requested URL in the session and redirects the user to the authentication form for the required factor. After authentication is successful, the browser is redirected back to the original requested URL. If the redirected request requires authentication of another factor, the authentication process for that factor starts. This process is repeated until all of the required factors are authenticated. Then the browser is allowed to complete their original request for a web page from the upstream server.

In the current implementation of MFAProxy, the redirection works best with HTTP GET requests, because no data can be lost. Some data may be lost if the proxy authentication process interrupts an HTTP POST request.

MFAProxy supports websockets and AJAX requests with limitations. The proxy cannot properly timeout websockets after they are established. Until the browser makes another request that has not been upgraded to support websockets, the connection will remain open. If a browser makes an AJAX call through the proxy that requires authentication, the proxy will respond with an HTTP 500 error. Single page web applications must respond appropriately by reloading the current page, so the proxy can authenticate the required factors and then redirect the browser back to the application's single page.

Passwords are implemented in MFAProxy using a local file in Apache htpasswd format. The htpasswd utility should be used to manage users and passwords in this file. The passwords must be encrypted with bcrypt. This encryption algorithm is used, because it slows down brute force attacks in the event the proxy's password database is compromised.

MFAProxy supports HOTP and TOTP using a common shared secrets file in a colon delimited text format similar to the htpasswd file. The file contains one line per user. The username is the first value and the shared secret is the second value. The shared secret for each user is created by base32 encoding 80 bits of pseudo-random data. The following command can be used to generate an appropriate shared secret on a Unix-like operating system: "head -c 10 /dev/urandom — base32". The proxy currently only stores one shared secret per user, so each user can only have one HOTP authenticator and one TOTP authenticator, and they must be based on the same shared secret.

To support HOTP the proxy maintains a counter for every user that successfully authenticates an HOTP factor. These counts are stored in a directory that contains one file for each user. Since the user's counter may race ahead of the proxy's counter, MFAProxy checks the next 20 possible counter values. If any of those values match, the factor is considered authenticated, and the server's counter file is updated with the corresponding value. If the user's counter gets more than 20 ahead of the proxy, the counter value on the server will need to be re-synchronized manually by editing the counter file directly on the proxy.

The proxy may be able to detect if the user has cloned their HOTP token. The proxy checks the 20 counter values lower than the current counter. If any of these counter values can be validated, the factor is not authenticated and the possible cloned token is reported in the proxy's log.

To support TOTP the proxy's clock should be synchronized with a clock source that is accurate to within a few seconds. MFAProxy will check the previous interval, the current interval, and the next interval. If any of these intervals can be validated, the factor will be authenticated.

Because TOTP relies only on the current time, MFAProxy should not need to store any state to authenticate a TOTP token, however it must store the interval of the most recent successful TOTP code for each user to prevent a specific type of replay attack. TOTP codes are valid for

approximately 30 seconds. An attacker that can capture a user's authentication code in real-time will also be able to reuse that code for approximately 30 seconds. The proxy compares the previous, next, and current intervals to the last successful interval for each user. A code can only be used if its corresponding interval is greater than the last successful interval. This prevents codes from being used more than once, so the proxy can prevent this type of replay attack. This also effectively limits the rate of successful TOTP authentications to no more than one every 30 seconds.

To support WebAuthn tokens the proxy must provide web pages and JavaScript handlers for registration and login using the proxy's reserved host name, because this name is embedded in all WebAuthn requests to the proxy. Registering a token with a site creates a unique identifier that associates the token with that specific site. MFAProxy uses the same token registration to authenticate access to multiple upstream sites, because all authentication requests from the proxy originate from the reserved host name.

WebAuthn registration uses the workflow shown in figure 3.1, and login uses the workflow shown in figure 3.2. Each process uses an initial AJAX GET request from the web browser to the proxy, followed by an AJAX POST request from the browser to the proxy. The JavaScript that handles the POST requests must be explicitly executed synchronously, or the POST will be interrupted by the browser processing the response to the initial GET.

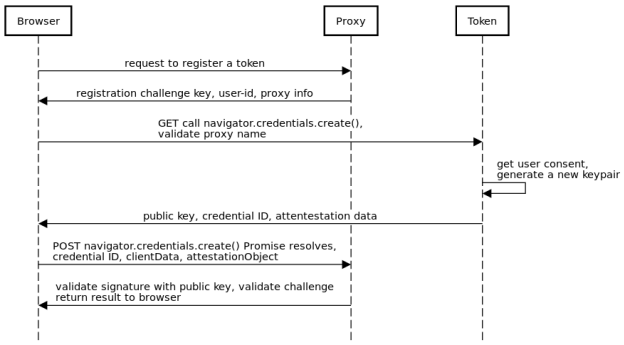

Figure 3.1 WebAuthn Registration

Tokens cannot be registered directly on the proxy host system, because the token must communicate with the proxy through the WebAuthn client built-in to a supported web browser. Only one WebAuthn token is supported per user. If a user does not have a WebAuthn token registered,

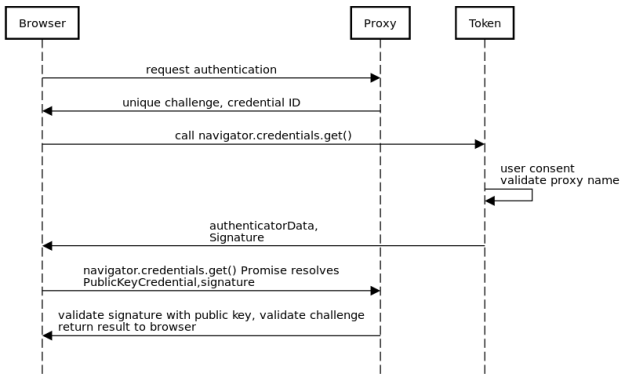

Figure 3.2 WebAuthn Login

they are only be allowed to register a token. If a user already has a WebAuthn token registered, they are only allowed to login with their existing token. Passwords should always be required by MFAProxy, when WebAuthn is used to prevent anonymous users from registering their own tokens with the proxy.

The proxy does not store any shared secrets for WebAuthn factors. It stores an ID and a public key that is unique to the combination of proxy, token, and registration. These IDs and keys cannot be used to access any other site. Because the proxy does not have access to the private key on the token, the information stored on the proxy cannot be used to clone WebAuthn tokens.

As with HOTP, the proxy stores a counter with every token registration. If an attacker had the resources to clone a WebAuthn token, this counter could be used to detect it. If a token sends a counter that is lower than the server's counter, the server will not validate the WebAuthn factor, and it will log the failed request as a possible cloned token. Because the counter is not otherwise used in the validation of the token, there are no re-synchronization issues if the token's counter races far ahead of the server's counter.

CHAPTER 4. DEPLOYMENT

When deploying any security software like MFAProxy, an organization should consider their security goals, end user requirements, and available resources. The proxy can be deployed on the network in a number of different configurations to achieve varying levels of security and functionality. Any implementation of multi-factor authentication will impact system usability. Hardware or software will need to be supplied and configured by the organization or its end users to use HOTP, TOTP, or WebAuthn tokens.

4.1 Server Deployment

Because MFAProxy acts as a general purpose reverse HTTP proxy, it can be deployed at different locations within a network to achieve different security goals. Two example configurations are presented here: a bastion host for multiple upstream servers and a local proxy for a single server.

The network diagram in figure 4.1 shows MFAProxy configured as a front-end server for multiple upstream servers. The proxy runs on a system at the perimeter of the network and has external and internal network interfaces that separate the two security domains. Users of the upstream servers can be internal or external to the local network. This configuration relies on split internal and external DNS.

Internal users access the upstream servers directly using each upstream server's IP address returned by an internal DNS server. Their requests are not routed through the proxy, so they are not required to use multi-factor authentication. These users experience no changes in usability after the deployment of MFAProxy.

External users access the upstream servers through MFAProxy using the IP address of the proxy's external interface returned by an external DNS server. Their requests are routed through

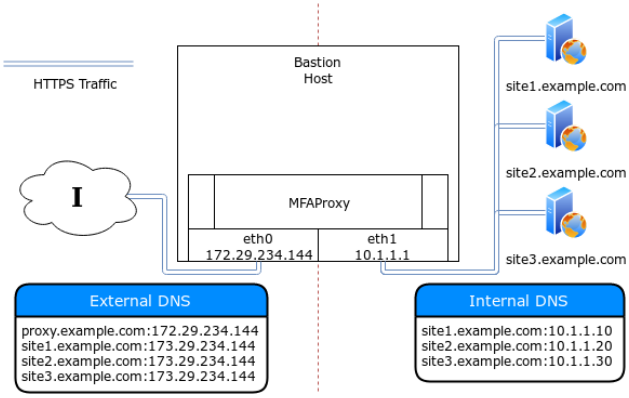

Figure 4.1 Bastion Host

the proxy, so they are required to authenticate all of the factors required by the proxy for each upstream server. The same factors can be used to authenticate access to multiple upstream servers, so external users may only need to authenticate once per session for each required factor.

The authentication timers set on the pattern for each upstream server dictate how frequently an external user will be required to re-authenticate each factor. An organization should balance their requirements for security with the need for usability. A password factor may use a long duration timer, because it requires the user to manually enter something they know that could be quite complex. An HOTP or TOTP factor may use a slightly shorter duration timer, because it only requires the user to enter a six digit code. A WebAuthn factor may use the shortest timer, because it only requires the user to push a button on a token.

MFAProxy configured as a bastion host serves a similar role as a virtual private network (VPN). It requires that users from outside the network provide an additional level of authentication, before they can access a restricted set of resources on the internal network. Like a VPN the proxy requires encryption between the browser and the proxy, even if the upstream servers do not support HTTPS. Unlike a VPN, traffic between the browser and the proxy is not encapsulated in an additional layer of encryption. A VPN can tunnel nearly any type of traffic, while the proxy is limited to HTTPS traffic.

The network diagram in figure 4.2 shows MFAProxy configured as a front-end server for a single upstream web server running on the same host as the proxy. The upstream site is bound to the

loop-back interface on the host, so it is only accessible locally on the host itself. MFAProxy is bound to the host's network interface, and the upstream server is set to forward requests to the site bound to the loop-back interface.

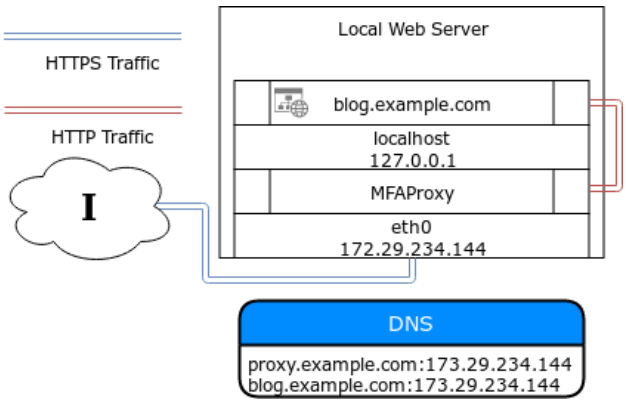

Figure 4.2 Local Host

In this configuration all users are in the same security domain, and all are required to authenticate the factors required by the proxy. The upstream web server does not need to support HTTPS to maintain confidentiality, because it can be provided by the proxy, and the upstream clear text traffic will not traverse the network. A configuration that requires multi-factor authentication for all users reduces the risk of phishing attacks regardless of the user's location on the network.

The authentication timers set on the pattern for the single local upstream server provide the same balance of usability and security as the bastion host example. The only additional consideration is that all users will always be subjected to the additional authentication requirements of the proxy.

The patterns that MFAProxy uses to match web requests can include part of the URL path starting at the root of the site. This makes it possible to require more frequent authentication or additional factors for specific parts of a site that require higher security. Using a configuration similar to the single site example, most parts of a site could use the proxy only to enable support for HTTPS with no additional authentication required besides the site's own passwords. A specific pattern could be defined for URLs starting with /admin that requires a second factor to access the administrative features of a site.

4.2 Password and Token Deployment

All authentication factors require some type of deployment. The way an organization deploys these factors may be just as important as the factors they chose to use. Each type of factor provides a mix of control by the user and by the organization that can vary depending on how the organization decides to deploy them.

As covered in the introduction, passwords have widely known and well understood weaknesses, but they are still the primary type of factor based on what the user knows. There is not much an organization can do to control what a user does with their password, but that does not mean there are no password deployment issues to consider.

In MFAProxy passwords are set by the administrator and cannot be changed by the end user. This limitation allows the administrator to require strong passwords. An organization should consider deploying password managers for its end users, so they will not need to memorize a large collection of strong passwords.

When used with a web browser, many password managers provide an additional benefit. They offer to automatically fill-in passwords by matching entries stored in their database with the host name in the URL of the current page. This can prevent users from providing their password to phishing sites that use a similar host name or similar appearance to deceive users. The feature is more valuable when passwords are used with HOTP and TOTP tokens, because these factors do not provide any verification of the host name.

If an upstream site already includes password authentication that provides enough security, MFAProxy can be configured not to require its own passwords by setting the password timer on a pattern to zero. MFAProxy can be configured to only enforce HOTP, TOTP, or WebAuthn token authentication as a second factor, while the site relies on its own passwords for the first factor.

HOTP is generally considered less secure than TOTP, because HOTP codes are valid until they are used, while TOTP codes change at a regular interval. If users are given their shared secret, this is an important concern. The user can redeploy that secret on any token they want without any oversight.

If an organization deploys HOTP using a physical token that prevents the user from easily reading the shared secret, it becomes much more difficult for the user to clone their shared secret onto an insecure token. If the organization also selects a token that emulates a keyboard rather than shows the code on a display, the length of time the code is valid is much less of a concern, because it would normally only be visible at the time the code is entered.

TOTP is often deployed when an organization expects end users to provide their own tokens and install the shared secrets themselves. The most common form of TOTP token is an app running on a mobile device. The organization has no control over what these apps do with the shared secrets, so organizations are advised to recommend apps that have been audited and are known to follow strong security practices. The time-limited nature of TOTP codes makes the display of the codes less of a concern. Instead organizations should pay close attention to the storage and backup polices of the apps they recommend, because the shared secrets are the equivalent of plain text.

Some TOTP tokens, especially those integrated into password managers, allow shared secrets to be synchronized on multiple devices. This may be beneficial in some deployments, because it enables multiple devices to be used as backup tokens with only a single shared secret.

U2F and WebAuthn provide a much higher level of security than tokens based on one-time passwords, because they rely on public-key cryptography instead of cryptographic hashes. They also have the most restrictive deployment options. Only tokens that are compatible with U2F or WebAuthn can be used. Currently that means only hardware tokens from a relatively small number of manufacturers like Yubico and Google are compatible.

No additional services need to be deployed to protect U2F and WebAuthn tokens from phishing attacks. Because the site's host name is embedded in the registration and login requests, these tokens cannot be fooled by similar looking sites or host names.

Each token must be registered individually on sites that require it. There are currently no options to rapidly deploy a large number of these types of tokens. As a result many organizations may provide their end users with compatible tokens, but they will leave it up to their users to register them.

CHAPTER 5. POTENTIAL VULNERABILITIES AND WEAKNESSES

MFAProxy is intended to increase security by enforcing multi-factor authentication on the sites it serves. Because the proxy adds to the attack of surface of a site, any vulnerabilities in the proxy itself have the potential to reduce or eliminate the additional security provided by the proxy. Any of these vulnerabilities could also allow an attacker to compromise the security of the proxy's host system.

The proxy can reduce but not eliminate the threats from certain types of attacks. Some of these limitations are inherent in the HTTP protocol or in the authentication factors. Others are the result of the design of the proxy itself.

Multi-factor authentication is often promoted as the solution to phishing attacks. While it certainly reduces the risks of these types of attack, it cannot eliminate them entirely. All types of multi-factor authentication are effective against simple phishing sites that gather credentials for reuse at a later time, but some factors are still vulnerable to Man-In-The-Middle (MITM) attacks.

Tools like Modlishka Duszyski (2019a) are available to run real-time MITM attacks on sites that support multi-factor authentication. It acts as a reverse proxy that captures usernames, passwords, and authentication codes as they are submitted. There are already examples of tools like this being used in the wild to take over accounts of YouTube "influencers" that had secured their accounts with TOTP tokens Vaas et al. (2019).

Real-time attacks on TOTP depend on the 30 second interval that an authentication code remains valid, which allows the code to be reused by an attacker within the same interval. MFAProxy reduces the risk of this type of attack by preventing TOTP codes from being reused.

This MITM attack also depends on getting the user's browser to access the target site through the attacker's reverse proxy instead. Duszyski (2019b) developed a technique called Client Domain Hooking that can redirect a web browser from the desired site to the attacker's site. It depends

on the fact that many browsers default to the clear-text HTTP protocol which can be intercepted and modified by the attacker using ARP cache poisoning, DNS cache poisoning, XSS injection, and malicious URLs.

MFAProxy reduces the risk of this type of MITM attack by enforcing HTTP Strict Transport Security (HSTS) for all requests handled by the proxy. This ensures that all subsequent access to these URLs will only use the TLS encrypted HTTPS protocol. This cannot completely eliminate the risk of this type of attack, because the browser must first connect to the proxy to populate its HSTS database. Every browser is at risk during its initial connection to each host name, if it tries an HTTP connection first.

Multi-factor authentication based on WebAuthn can eliminate all threats of a MITM attack, because it includes the host name in every authentication request. If a user tried to login to a site through this type of reverse proxy with a WebAuthn token, the token simply would not respond to the challenge appropriately.

Regardless of the type of authentication used, MFAProxy is vulnerable to some types of session hijacking. In this attack a user authenticates with the proxy successfully, and then the attacker copies the session identifier onto another system. MFAProxy does take steps to reduce the risk of session hijacking and session fixation.

The proxy uses a random 256-bit session ID, so the identifier cannot be easily guessed. The identifier is updated each time a user successfully authenticates another factor. It is deleted and not reused after the user logs out of the proxy. The session manager enforces an inactivity timer and a maximum lifetime for every session.

The cookie that stores the session is not persistent, so the browser should not store the session identifier locally on disk. The cookie also specifies the HTTP-only flag, so the session identifier is not available to JavaScript through the browser's document-object model. This prevents JavaScript-based malware from obtaining the session ID and sending it to a third-party site.

MFAProxy is vulnerable to Trojans and any other type of malware running on the end-user's system that can read the session ID from the browser's memory. This could be mitigated somewhat

by associating a session ID with a source IP address and enforcing that source address for all requests. This mitigation would impact usability for some users that connect through firewalls and VPNs that utilize multiple source IP addresses.

Because MFAProxy is not integrated directly into the web applications it protects, it is possible for an attacker to bypass the proxy by establishing connections directly to an upstream site. If any system that has direct access to the upstream site is compromised, that system can be used to bypass the proxy. In cases where the upstream server is bound to the loop-back interface on the proxy system, any local non-privileged user account on that system can still connect directly to that server. An attacker could setup remote port forwarding through an outbound SSH connection to make a local server available on a remote system and bypass the proxy.

Proxy bypass can be mitigated by limiting user accounts on servers as much as possible and ensuring that all accounts on that system are properly secured. An organization may also want to restrict outbound connections from servers to prevent anyone from forwarding traffic back through those connections.

CHAPTER 6. FUTURE WORK

While MFAProxy is able to add support for multi-factor authentication to sites that do not currently support it, additional features would make the proxy more useful in a production environment.

6.1 Multiple tokens per factor type

The proxy only supports one shared secret and one WebAuthn token registration per user. Users should be able to register more than one token of each type. Many users would want to be able to register a primary token in their mobile device and a backup token in a physical hardware device in case they lost their phone.

This would require a one-to-many mapping between users and secrets and users and WebAuthn authenticators. Each token would need to be associated with its own counter or its last successful TOTP authentication code. As this data model gets more complex, it would be appropriate to migrate the data from simple text files to a small relational database.

6.2 Remember this device

Some systems allow users to authenticate their browsers for an extended period of time by supporting a "remember this device" option. This stores a persistent cookie in the browser that marks it as a recognized device for an extended period of time after an authentication factor has been validated.

6.3 Backup codes

Most multi-factor authentication systems enable the user to generate a number of backup codes that allow the user to access the site in the event they lose a token. The proxy would need to

restrict the use of each backup code to a single factor for single session, so they cannot be used as the basis for replay attacks.

6.4 User and site management

Currently user accounts are managed through a series of text files on the proxy that must be kept synchronized with each other. Providing a web interface to manage and audit accounts and sites would allow the proxy administrator to easily determine which factors are enabled for each user, and which factors are required by each site.

6.5 Session management

MFAProxy provides a status page that shows each user when they authenticated each factor in their current session. A proxy-wide session status page for administrators would allow them to view currently active sessions and authenticated factors. It could also allow the administrator to terminate active sessions that violate the organization's security policy.

6.6 Device attestation

At the present time MFAProxy does not support device attestation. This prevents WebAuthn tokens from being used as multiple factors on their own. As support for WebAuthn becomes more widespread, the proxy will need to either attest specific devices using a hard-coded certificate store or check a central certificate authority to attest devices. If a device can be attested, and it supports multi-factor authentication, administrators may want to allow users to authenticate to the proxy using these attested devices on their own to support password-less authentication.

CHAPTER 7. CONCLUSION

Information security is about managing information-related risks. Sites that rely only on a single factor for authentication are at great risk of being compromised from weak passwords, password reuse, and phishing attacks. Multi-factor authentication can reduce these risks, and MFAProxy can provide multi-factor authentication on sites that do not natively support it.

MFAProxy supports a flexible deployment within a network to meet specific security goals. It can be used on the perimeter of a network to protect sites from external attacks. It can be used directly on a web server to protect a site from compromised systems within an organization.

The proxy supports a variety of tokens that offer different security and usability profiles. Organizations should consider the security benefits and the requirements of each token type, and then develop a deployment plan that meets their security goals. It is important to consider more than the complexity of the authentication codes and the data that gets sent over the network. Some factors give more control to the organization and others give more control to the end user. End users may use their tokens in ways that organizations do not intend, and device attestation continues to be difficult to enforce.

MFAProxy can reduce but not eliminate the risks associated with passwords and single-factor authentication. The risk of Man-In-The-Middle attacks are particularly difficult to mitigate, but all forms of multi-factor authentication provide much better security than passwords alone.

BIBLIOGRAPHY

Akers, A. (2019). Sms intercept attacks and why sms multi-factor still matters. Available: https://auth0.com/blog/why-sms-multi-factor-still-matters/. [Online; accessed 8-October-2019].

Alliance, F. (2019). Download specifications. Available: https://fidoalliance.org/specifications/download/. [Online; accessed 8-October-2019].

Constantin, L. (2019). Password managers remain an important security tool despite new vulnerability report. Available: https://www.csoonline.com/article/3344298/password-managers-remain-an-important-security-tool-despite-new-vulnerability-report.html. [Online; accessed 8-October-2019].

Duszyski, P. (2019a). drk1wi/modlishka. Available: https://github.com/drk1wi/Modlishka. [Online; accessed 8-October-2019].

Duszyski, P. (2019b). Hijacking browser tls traffic through client domain hooking. Available: https://github.com/drk1wi/assets/raw/master/Hijacking%20browser%20TLS%20traffic%20through%20Client%20Domain%20Hooking%20-%20Piotr%20Duszynski.pdf. [Online; accessed 8-October-2019].

Edwards, A. (2019). alexedwards/scs. Available: https://github.com/alexedwards/scs. [Online; accessed 8-October-2019].

Foundation, T. O. (2019). Owasp/cheatsheetseries. Available: https://github.com/OWASP/CheatSheetSeries/blob/master/cheatsheets/Session_Management_Cheat_Sheet.md. [Online; accessed 8-October-2019].

Hunt, T. (2019). Have i been pwned? Available: https://haveibeenpwned.com/. [Online; accessed 8-October-2019].

M'Raihi, D., Bellare, M., Hoornaert, F., Naccache, D., and Ranen, O. (2005). Hotp: An hmac-based one-time password algorithm. RFC 4226, RFC Editor. Available: http://www.rfc-editor.org/rfc/rfc4226.txt. [Online; accessed 8-October-2019].

M'Raihi, D., Machani, S., Pei, M., and Rydell, J. (2011). Totp: Time-based one-time password algorithm. RFC 6238, RFC Editor. Available: http://www.rfc-editor.org/rfc/rfc6238.txt. [Online; accessed 8-October-2019].

Vaas, L., Lee, R., and Varmazis, M. (2019). Youtube 'influencers' get 2fa tokens phished. Available: https://nakedsecurity.sophos.com/2019/09/24/youtube-influencers-get-2fa-tokens-phished/. [Online; accessed 8-October-2019].

Yıldırım, M. and Mackie, I. (2019). Encouraging users to improve password security and memorability. *International Journal of Information Security*, pages 1–19.